УДК 519.673, 004.451.46

ОПТИМИЗАЦИЯ ПРОИЗВОДИТЕЛЬНОСТИ ВЫЧИСЛЕНИЙ ДЛЯ МОДЕЛИРОВАНИЯ ЦУНАМИ НА ПАРАЛЛЕЛЬНЫХ АРХИТЕКТУРАХ

Курако М.А.
Научный руководитель – доктор технических наук Симонов К.В.

Сибирский федеральный университет

Работы по совершенствованию оперативного прогноза цунами с использованием глубоководных регистраторов активизировались после катастрофического цунами в Индийском океане 26 декабря 2004 года, которое привело к гибели более 250 тысяч человек.

Приближения теории мелкой воды (как линейное, так и нелинейное) используются как основные модели для описания распространения волны в океане. Эти модели достаточно точно отражают основные параметры волн (время распространения от очага до записывающего приемника и амплитуды) даже для достаточно грубой цифровой батиметрии в предположении, что начальное смещение дна в источнике известно.

Пакет программ MOST использует модель расчета распространения волны цунами над глубоководной акваторией при помощи метода расщепления по пространственным переменным. Для численного расчета распространения волны цунами используется нелинейная система дифференциальных уравнений мелкой воды. Модель мелкой воды хорошо описывает процесс распространения волн цунами в открытом океане при условии, что горизонтальные размеры подвижки океанического дна, генерирующие эту волну, значительно превосходят глубину океана.

Алгоритм численного решения системы строится на основе метода расщепления по пространственным направлениям. Для этого рассматриваются две вспомогательные системы, каждая из которых зависит только от одной пространственной переменной. В дальнейшем для численного решения системы достаточно построить устойчивые разностные схемы.

Расчет движения волны происходит в два этапа: на первом шаге производится вычисление смещения волны вдоль оси X, на втором – вдоль оси Y. При этом расчет вдоль разных строк данных по оси X можно производить независимо. Аналогичная ситуация и для расчета вдоль оси Y.

Для расчета используются четыре матрицы: глубина, высота волны, скорость движения волны вдоль оси X и скорость движения волны вдоль оси Y. Размерность матриц равна размерности расчетного поля и для акватории Тихого океана составляет 2581x2879. Далее представлена реализация программного комплекса MOST на параллельных архитектурах.

Для реализации расчетов использованы многоядерные вычислительные системы и графические процессоры. Многоядерные системы являются системами с общей памятью, что позволяет использовать их преимущества в скорости обмена данными между ядрами, с другой стороны, такие системы обладают меньшей масштабируемостью. Для графических процессоров характерны высокий уровень параллелизма, заложенный в архитектуру (сотни независимых аппаратных потоков) и различные способы работы с памятью, позволяющие строить эффективные схемы

обменов между памятью хоста и собственной памятью, уменьшая тем самым влияние кэш-памяти и маскируя латентность памяти.

Наиболее эффективным способом распараллеливания вычислительных алгоритмов для систем с общей памятью является применение технологии OpenMP. Этот подход ориентирован на распределение итераций циклов между доступными программе процессорами (ядрами). При этом рекомендуется с помощью OpenMP производить распараллеливание внешних циклов. Такими циклами в программном комплексе MOST являются циклы расчетов смещения волны вдоль осей X и Y. Разработанный программный модуль интегрируется в исходные коды программы MOST путем замены вычислительной части. Тестирование проводилось на ряде вычислительных узлов, предоставленных компаниями SUN Microsystems и Intel (рис. 1).

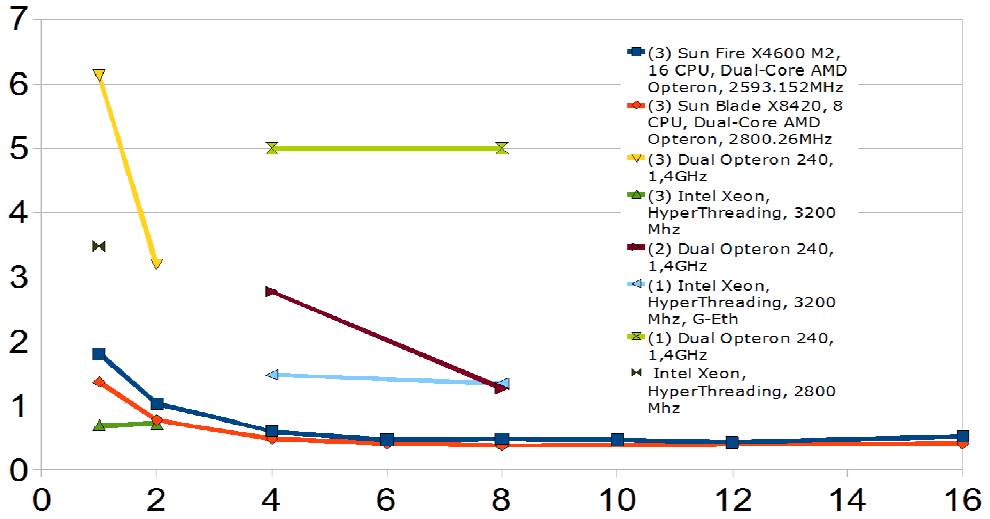

Рис. 1. Зависимость времени вычисления одной итерации от количества потоков. По горизонтали – количество потоков, по вертикали – время вычисления одной итерации в секундах. Размер области моделирования – 2581x2879 точек

Количество итераций основного цикла может составлять около 9000, что соответствует примерно 24 часам реального времени, которое требуется для распространения волны по всему океану. Все итерации содержат примерно равное количество операций, поэтому показателем ускорения может служить время выполнения одной итерации.

Анализ этой диаграммы (рис. 1) показывает, что данный подход к распараллеливанию целесообразно применять для систем с количеством процессоров не более шести. Хорошая масштабируемость достигается для систем с числом вычислительных ядер до 4, после чего происходит падение производительности при использовании 8 и более вычислительных ядер. Это связано с тем, что итерации цикла выполняются очень быстро и процессорам (вычислительным ядрам) требуется часто обращаться к общей области памяти, что приводит к необходимости производить синхронизацию кэшей, на что и тратится основное время.

Один из стандартных подходов к адаптации программ под GPU (графические процессоры) – это последовательный перенос участков кода на GPU. Вычисления параметров волны вдоль оси X, а затем вдоль оси Y могут производиться независимо для каждой строки/столбца данных. Более того, можно независимо вычислять значения инвариантов и новых значений высоты волны и скоростей вдоль осей для каждой точки

пространства. Благодаря высокой степени параллелизма, заложенного в данном алгоритме, возможна его эффективная реализация для графических процессоров.

В рамках реализации этого подхода разработан соответствующий код. Размер блока потоков взят равным 16x16 потоков. Таким образом, все пространство моделирования разбито на равные блоки. Как в дальнейшем показало профилирование и вычислительные эксперименты, именно такая конфигурация является оптимальной для загрузки потоковых мультипроцессоров GPU. При этом на видеокарте NVIDIA GeForce 9800 GX2 время выполнения одной итерации составило в среднем 0.25 секунды.

Наиболее сложной в плане реализации оказалась функция расчета высоты волны. Внутри нее требовалось проводить большое количество проверок на граничные условия (наличие берегов, материков и островов) и вести интенсивное чтение данных из памяти. Для упрощения этой функции заранее выполнен просчет некоторых условий, что позволило исключить вложенные проверки. Кроме того, все необходимые для расчета данные перенесены в разделяемую память.

При моделировании выяснилось, что вычисление инвариантов вдоль осей X и Y происходит с некоторой неточностью, что определяется наличием в ядрах вызова функции вычисления квадратного корня, которая дает ошибку в 1-2 младших битах, в результате это приводит к самопроизвольному раскачиванию поверхности океана уже на 50 итерациях. Для решения этой проблемы рассмотрено несколько решений: переход на вычисления инвариантов с использованием чисел двойной точности, использование представления чисел двойной точности с помощью двух чисел одинарной точности, отказ от вычисления квадратного корня внутри основного цикла. Первые два варианта не приводят к большому снижению производительности. Последний вариант – отказ от вычисления квадратного корня – оказался наиболее простым и подходящим в плане реализации. Для того чтобы избавиться от постоянного вычисления квадратного корня решено производить все вычисления в инвариантах и только на стадии сохранения данных моделирования производить пересчет инвариантов назад к значениям высоты волны и ее скоростям по направлениям.

В результате модификации кода поверхность океана перестала самопроизвольно раскачиваться, время вычисления одной итерации уменьшилось и составило в среднем 0.23 сек на одну итерацию по времени. Еще на 10% удалось ускорить программу заменой в ядре, вычисляющем высоту волны, некоторых операций деления на умножение. В соответствие с документацией деление занимает в 9 раз больше тактов, чем умножение: 36 против 4 тактов. Тем самым время вычисления одной итерации составило 0.21 секунду.

Проверены несколько путей оптимизации программы:

– создание двух функций вычисления высоты волны, которые будут производить вычисления вдоль разных направлений без использования функции транспонирования матриц;

– выравнивание начала строк всех матриц в памяти GPU для того, чтобы получить последовательное когерентное чтение;

– переход к использованию текстурной памяти.

Как видно из графика (рис. 2), функция транспонирования матриц занимает около 26% времени выполнения программы, но используется только для того, чтобы создать удобное расположение данных в памяти для функции вычисления высоты волны на второй половине итерации. Это оправдано для последовательной версии программы и программы на OpenMP, поскольку это приводит к последовательному чтению из памяти и меньшему количеству кэш–промахов, но является лишним для реализации на GPU с учетом дальнейших оптимизаций. Таким образом, создана вторая

версия функции вычисления высоты волны, которая выполняет те же вычисления, но вдоль другой оси. В результате время выполнения одного цикла по времени сократилось до 0.135 секунд.

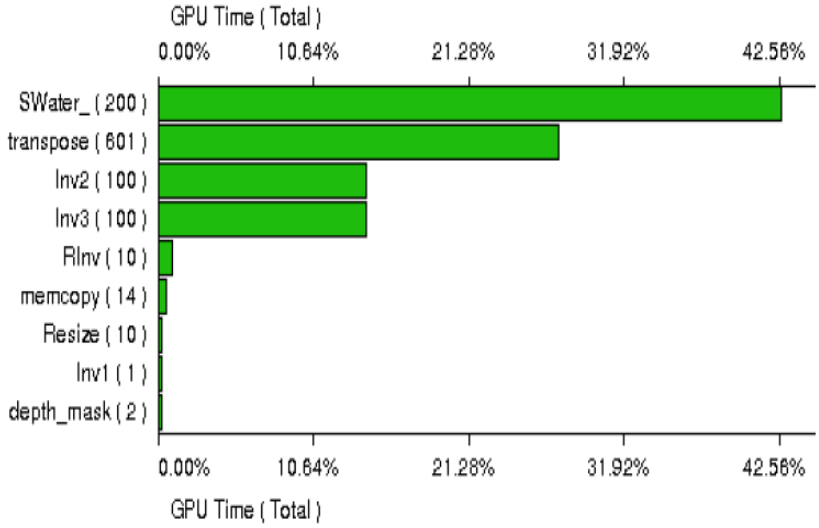

Рис. 2. Результаты замера времени выполнения функций

Для выявления направлений дальнейшей оптимизации использовался профилировщик CUDA. Анализ замеров показал, что несмотря на простой вид функций Inv2 и Inv3, вычисляющих инварианты вдоль осей, непоследовательное чтение и запись происходят по 2.6 миллиона раз, в то время как последовательное – всего несколько тысяч. Для того, чтобы получить последовательное чтение, произведена замена функций выделения памяти на их аналоги, производящие выделение выровненных участков памяти. В результате количество непоследовательных обращений к памяти сократилось до нуля и время обработки одной итерации по времени составило 0.037 секунды, что в 100 раз быстрее, чем для последовательной программы и в 14 раз быстрее, чем на 8 ядрах версии программы на OpenMP.

Для программ на CPU работа с памятью происходит эффективно, если данные в памяти расположены рядом друг с другом и легко помещаются в кэш или CPU может определить шаблон доступа к памяти, опять же заранее подгружая данные в кэш. У GPU для помещения данных в кэш можно использовать текстуры. При этом в кэш текстуры помещаются те данные, которые локализованы в двухмерном пространстве относительно данных, к которым идет обращение.

Дальнейшая оптимизация привела к использованию текстурной памяти и небольшой модификации вычислительных ядер. Выигрыш от такой оптимизации составил в среднем 0.03 секунды на итерацию – около 10%.

Для финального тестирования программа была выполнена на NVIDIA Tesla C1060. Полученный результат — 0.02 секунды на итерацию, что составляет итоговое ускорение около 170 раз по сравнению с исходной последовательной программой.

Таким образом, реализация программного комплекса MOST на параллельных архитектурах позволяет значительно увеличить производительность вычислений, уменьшая время вычислений с нескольких часов до нескольких минут. При этом применяемые оптимизации позволяют увеличить производительность OpenMP версии комплекса на порядок, а версии для графических процессоров – на 2 порядка. Исследования выполнены при тесном сотрудничестве с ведущими сотрудниками НГУ в этой предметной области: д.ф.-м.н. М.М. Лаврентьевым и к.т.н. А.А. Романенко.